OPEN

Vulnerable to TOCTOU issues

Sean Barnum, Cigital, Inc. [vita[1]]

2007-04-02

Part "Original Cigital Coding Rule in XML"

Mime-type: text/xml, size: 12433 bytes

<table>
<tr><td>Attack Category</td><td>Path spoofing or confusion problem</td></tr>
<tr><td>Vulnerability Category</td><td>Indeterminate File/Path
TOCTOU - Time of Check, Time of Use</td></tr>
<tr><td>Software Context</td><td>File Management
File I/O</td></tr>
<tr><td>Location</td><td>fcntl.h</td></tr>
<tr><td>Description</td><td> The open function establishes a connection between a file and a file descriptor. Pathname is the name of the file to open and fileFlags is the bitwise OR of a series of constants used to specify the file access modes. An optional additional input is used to specify the permissions, such as read-only. open() is vulnerable to TOCTOU attacks. A call to open() should be flagged if the first argument (the directory or file name) is used earlier in a check-category call. </td></tr>
<tr><td>APIs</td><td><table>
<tr><th>Function Name</th><th>Comments</th></tr>
<tr><td>_open</td><td>use; win32</td></tr>
<tr><td>_wopen</td><td>use; win32</td></tr>
<tr><td>open</td><td>use</td></tr>
<tr><td>fopen</td><td></td></tr>
<tr><td>_tfopen</td><td>Equivalent to fopen on Windows</td></tr>
<tr><td>_wfopen</td><td>Equivalent to fopen on Windows</td></tr>
</table></td></tr>
<tr><td>Method of Attack</td><td>The key issue with respect to TOCTOU vulnerabilities is that programs make assumptions about atomicity of actions. It is assumed that checking the state or identity of a targeted resource followed by an action on that resource is all one action. In reality, there is a period of time between the check and the use that allows either an attacker to</td></tr>
</table>

1. http://buildsecurityin.us-cert.gov/bsi-rules/35-BSI.html (Barnum, Sean)

	intentionally or another interleaved process or thread to unintentionally change the state of the targeted resource and yield unexpected and undesired results. The open() call is a use-category call, which when preceded by a check-category call can be indicative of a TOCTOU vulnerability. A TOCTOU attack in regards to open() can occur when a. A check for the existence of the file occurs or a non-fd reference (pathname) to the filename occurs b. An actual call to open occurs. Between a and b, an attacker could, for example, link the referenced file to a known file. The subsequent freopen() call would have an unintended effect or impact.
Exception Criteria	
Solutions	<table><tr><th>Solution Applicability</th><th>Solution Description</th><th>Solution Efficacy</th></tr><tr><td>Applies to most applications of open().</td><td>Consider using the safer set of steps outlined below for opening and creating files as outlined in Building Secure Software (referenced below), page 220. If this call must be used create a directory only accessible by the UID of the running program and only manipulate files in that directory. 1) Perform an lstat() of the file before opening it, saving the stat structure. 2) Perform Open(), passing the O_CREAT an O_EXCL</td><td>Effective.</td></tr></table>

		flags which will cause the open to fail if the file cannot be created. 3) Perform an fstat() on the file descriptor returned by the open() call, saving the stat structure. 4) Compare three fields in the two stat structures to be sure they are equivalent: st_mode, st_info & st_dev. If these comparisons are successful, then we know the lstat() call happened on the file we ultimately opened. Moreover, we know that we did not follow a symbolic link (which is why we used lstat() instead of stat()). (This solution comes from the book Building Secure Software referenced below)	
	Applies to most applications of open().	The most basic advice for TOCTOU vulnerabilities is to not perform a check before the use. This does not resolve the	Does not resolve the underlying vulnerability but limits the false sense of security given by the check.

	<table><tr><td></td><td>underlying issue of the execution of a function on a resource whose state and identity cannot be assured, but it does help to limit the false sense of security given by the check.</td><td></td></tr><tr><td>Applies to most applications of open().</td><td>Limit the interleaving of operations on files from multiple processes.</td><td>Does not eliminate the underlying vulnerability but can help make it more difficult to exploit.</td></tr><tr><td>Applies to most applications of open().</td><td>Limit the spread of time (cycles) between the check and use of a resource.</td><td>Does not eliminate the underlying vulnerability but can help make it more difficult to exploit.</td></tr><tr><td></td><td>Recheck the resource after the use call to verify that the action was taken appropriately.</td><td>Effective in some cases.</td></tr></table>
Signature Details	int open (const char *path , int oflag, /* mode_t mode */...) FILE *_tfopen(const wchar_t *filename, const wchar_t *mode) FILE *_wfopen(const wchar_t *filename, const wchar_t *mode) FILE *fopen(const char *filename, const char *mode)

Examples of Incorrect Code	<pre>/* Access check added */
	
	void die(char *msg)
	{
	perror(msg);
	exit(1);
	}</pre>

	``
	int main()
	{
	int fd;
	int access_stat;
	
	access_stat=access("testfile",
	R_OK);
	if (access_stat = F_OK)
	{
	if ((fd =
	open("testfile",O_CREAT|
	O_WRONLY,0)) < 0)
	die("open failed");
	if (write(fd,"output\n",7) < 0)
	die("write failed");
	if (close(fd) < 0)
	die("close failed");
	}
	return 0;
	}
	``

	``
	[...]
	FILE *fp;
	struct stat lstat_info;
	if (lstat(fname, &lstat_info) ==
	-1) {
	fp1 = fopen("test.txt", "w");
	} else {
	fp1 = fopen("test.txt", "r+");
	}
	[...]
	``

Examples of Corrected Code	```
	/** This example was taken from
	the book Building Secure Software
	**/
	/** Viega, John & McGraw, Gary.
	Building Secure Software: How to
	Avoid Security Problems the Right
	Way. Boston, MA: Addison-Wesley
	Professional, 2001, pg. 220 **/
	
	#include <sys/stat.h>
	#include <sys/types.h>
	#include <sys/stat.h>
	#include <fcntl.h>
	#include <unistd.h>
	#include <stdio.h>
	#include <errno.h>
	
	FILE *safe_open_wplus(char *fname)
	{
	struct stat lstat_info,
	fstat_info;
	FILE *fp;
	``

```
char *mode = "rb+"; /*We perform
our own truncation.*/
int fd;

if(lstat(fname, &lstat_info) == -1)
{
/* If the lstat() failed for
reasons other than the file not
existing, return 0, specifying
error. */
if( errno != ENOENT ) {
return 0;
}

if((fd = open(fname, O_CREAT|
O_EXCL| O_RDWR, 0600)) == -1) {
return 0;
}
mode = "wb";
} else {

/* Open an existing file */
if((fd = open(fname, O_RDWR)) ==
-1) {
return 0;
}

if(fstat(fd, &fstat_info) == -1 ||
lstat_info.st_mode !=
fstat_info.st_mode ||
lstat_info.st_ino !=
fstat_info.st_ino ||
lstat_info.st_dev !=
fstat_info.st_dev ) {

close(fd);
return 0;
}

/* Turn the file into an empty
file, to mimic w+ semantics. */
ftruncate(fd, 0);
}

/* Open a stdio file over the low-
level one */
fp = fdopen(fd, mode);
if(!fp) {
close(fd);
unlink(fname);
return 0;
}
return fp;
}
```

```
/* No check */

void die(char *msg)
```

```
{
perror(msg);
exit(1);
}

int main()
{
int fd;

if ((fd = open("testfile",O_CREAT|
O_WRONLY,0)) < 0)
die("open failed");
if (write(fd,"output\n",7) < 0)
die("write failed");
if (close(fd) < 0)
die("close failed");

return 0;
}
```

```
struct stat lstat_info,
fstat_info;
FILE *fp;
char *mode = "r+"
int fd;

if (lstat(fname, &lstat_info) ==
-1) {
if (errno != ENOENT) {
return 0;
}
if ((fd = open(fname, O_CREAT |
O_EXCL | O_RDWR, 0600)) == -1) {
return 0;
}
mode = "w";
}
else {
if ((fd = open(fname, O_RDWR)) ==
-1) {
return 0;
}

if (fstat(fd, &fstat_info) == -1 ||
lstat_info.st_mode !=
fstat_info.st_mode ||
lstat_info.st_ino !=
fstat_info.st_ino ||
lstat_info.st_dev !=
fstat_info.st_dev) {
close(fd);
return 0;
}
ftruncate(fd, 0);
}

fp = fopen(fd, mode);
```

	`if (!fp) {`
	`close(fd);`
	`unlink(fname);`
	`return 0;`
	`}`

Source References	• Viega, John & McGraw, Gary. *Building Secure Software: How to Avoid Security Problems the Right Way*. Boston, MA: Addison-Wesley Professional, 2001, ISBN: 020172152X, pg. 220 • UNIX man page for open() • Microsoft Developer Network Library (MSDN). • Re: gzip TOCTOU file-permissions vulnerability[2] (2005).
Recommended Resource	
Discriminant Set	**Operating Systems**: • UNIX • Windows **Languages**: • C • C++

Cigital, Inc. Copyright

The Build Security In (BSI) portal is sponsored by the U.S. Department of Homeland Security (DHS), National Cyber Security Division. The Software Engineering Institute (SEI) develops and operates BSI. DHS funding supports the publishing of all site content.

1. mailto:copyright@cigital.com